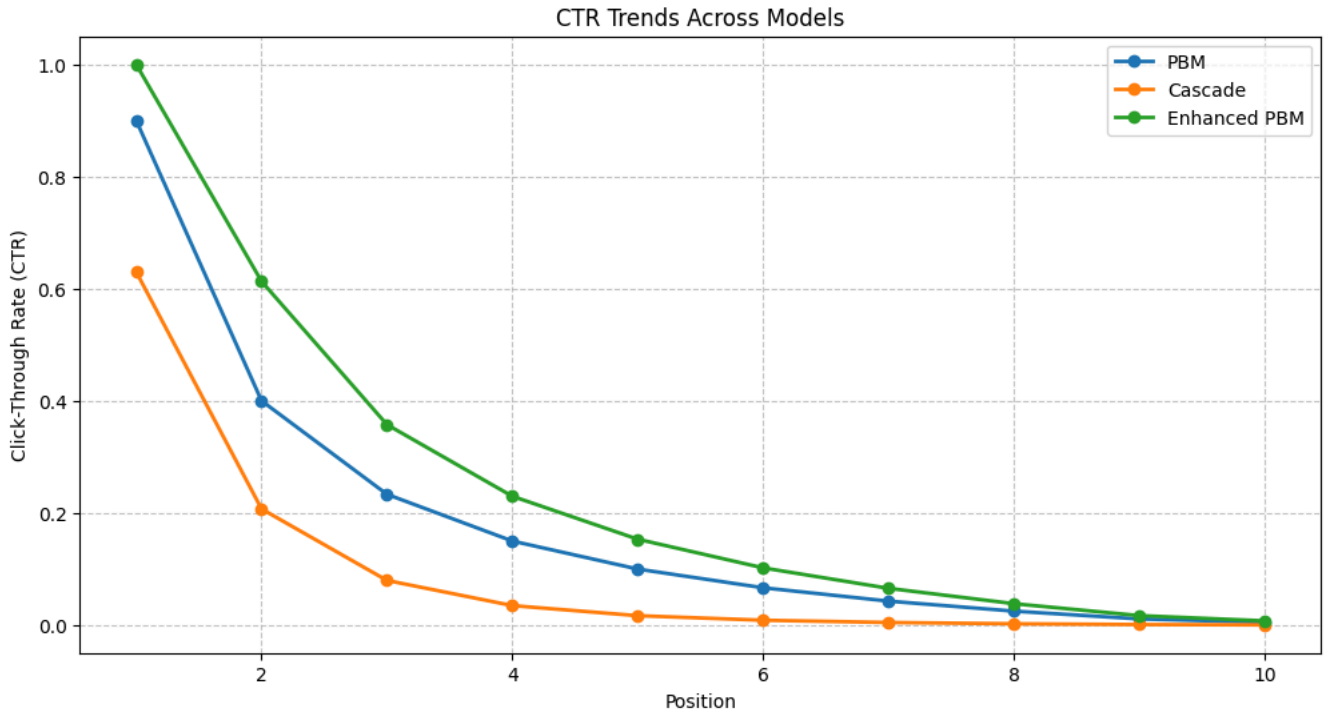
```
# Install required packages
!pip install numpy pandas matplotlib seaborn
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.13.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.55.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas)
```

```
# Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from typing import List, Tuple, Dict

class ClickModel:
    """Base class for click models."""
    def __init__(self):
        pass

    def predict_click_probabilities(self, positions: List[int], relevance_scores: List[float]) -> List[float]:
        raise NotImplementedError

class PositionBasedModel(ClickModel):
    def __init__(self, position_biases: List[float] = None):
        super().__init__()
        if position_biases is None:
            self.position_biases = [1.0 / (i + 1) for i in range(10)]
        else:
            self.position_biases = position_biases

    def predict_click_probabilities(self, positions: List[int], relevance_scores: List[float]) -> List[float]:
        click_probs = []
        for pos, rel in zip(positions, relevance_scores):
            bias = self.position_biases[pos] if pos < len(self.position_biases) else self.position_biases[-1]
            click_probs.append(bias * rel)
        return click_probs

class CascadeModel(ClickModel):
    def __init__(self, stopping_probs: List[float] = None):
        super().__init__()
        if stopping_probs is None:
            self.stopping_probs = [0.7 for _ in range(10)]
        else:
            self.stopping_probs = stopping_probs

    def predict_click_probabilities(self, positions: List[int], relevance_scores: List[float]) -> List[float]:
        click_probs = []
        continue_prob = 1.0

        for pos, rel in zip(positions, relevance_scores):
            stop_prob = self.stopping_probs[pos] if pos < len(self.stopping_probs) else self.stopping_probs[-1]
            click_prob = continue_prob * stop_prob * rel
            click_probs.append(click_prob)
            continue_prob *= (1 - stop_prob * rel)

        return click_probs

class EnhancedPBM(ClickModel):
    def __init__(self, position_biases: List[float] = None, query_weights: Dict[str, float] = None):
        super().__init__()
        if position_biases is None:
            self.position_biases = [1.0 / (i + 1) for i in range(10)]
        else:
            self.position_biases = position_biases

        if query_weights is None:
            self.query_weights = {
                "informational": 1.2,
                "navigational": 0.8,
                "transactional": 1.0
```

```
            }
        else:
            self.query_weights = query_weights

    def predict_click_probabilities(self, positions: List[int], relevance_scores: List[float],
                                    query_type: str = "informational",
                                    user_context: Dict = None) -> List[float]:
        query_weight = self.query_weights.get(query_type, 1.0)
        context_multiplier = 1.0

        if user_context:
            session_length = user_context.get("session_length", 1)
            context_multiplier *= min(1.2, 1.0 + 0.1 * np.log(session_length))

            hour = user_context.get("hour", 12)
            if 9 <= hour <= 17:
                context_multiplier *= 1.1

        click_probs = []
        for pos, rel in zip(positions, relevance_scores):
            bias = self.position_biases[pos] if pos < len(self.position_biases) else self.position_biases[-1]
            adjusted_relevance = rel * context_multiplier
            click_prob = bias * query_weight * adjusted_relevance
            click_probs.append(min(1.0, click_prob))

        return click_probs

# Evaluation function
def evaluate_model(model: ClickModel, test_data: List[Tuple[List[int], List[float], List[float]]],
                   metrics: List[str] = ["ndcg", "mse"]) -> Dict[str, float]:
    results = {}
    all_pred_probs = []
    all_actual_clicks = []

    for positions, relevance_scores, actual_clicks in test_data:
        pred_probs = model.predict_click_probabilities(positions, relevance_scores)
        all_pred_probs.extend(pred_probs)
        all_actual_clicks.extend(actual_clicks)

    if "mse" in metrics:
        mse = np.mean([(p - a) ** 2 for p, a in zip(all_pred_probs, all_actual_clicks)])
        results["mse"] = mse

    if "ndcg" in metrics:
        def dcg(scores):
            return sum((2**s - 1) / np.log2(i + 2) for i, s in enumerate(scores))

        pred_dcg = dcg(all_pred_probs)
        ideal_dcg = dcg(sorted(all_actual_clicks, reverse=True))
        ndcg = pred_dcg / ideal_dcg if ideal_dcg > 0 else 0
        results["ndcg"] = ndcg

    return results

# Visualization functions
def plot_ctr_trends(models_data: Dict[str, List[float]], save_path: str = None):
    plt.figure(figsize=(12, 6))
    positions = range(1, len(next(iter(models_data.values()))) + 1)

    for model_name, ctrs in models_data.items():
        plt.plot(positions, ctrs, marker='o', label=model_name, linewidth=2)

    plt.xlabel('Position')
    plt.ylabel('Click-Through Rate (CTR)')
    plt.title('CTR Trends Across Models')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()

    if save_path:
        plt.savefig(save_path)
    plt.show()

def plot_metrics_comparison(metrics_data: Dict[str, Dict[str, float]], save_path: str = None):
    metrics = list(next(iter(metrics_data.values())).keys())
    models = list(metrics_data.keys())

    fig, axes = plt.subplots(1, len(metrics), figsize=(15, 5))

    for i, metric in enumerate(metrics):
        values = [metrics_data[model][metric] for model in models]
        axes[i].bar(models, values)
        axes[i].set_title(f'{metric.upper()} Comparison')
        axes[i].set_xticklabels(models, rotation=45)
```

```
        plt.tight_layout()
        if save_path:
            plt.savefig(save_path)
        plt.show()

    if __name__ == "__main__":
        # Generate sample data
        positions = list(range(10))
        relevance_scores = [0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.05]

        # Initialize models
        pbm = PositionBasedModel()
        cascade = CascadeModel()
        enhanced_pbm = EnhancedPBM()

        # Get predictions
        pbm_probs = pbm.predict_click_probabilities(positions, relevance_scores)
        cascade_probs = cascade.predict_click_probabilities(positions, relevance_scores)
        enhanced_probs = enhanced_pbm.predict_click_probabilities(
            positions,
            relevance_scores,
            query_type="informational",
            user_context={"session_length": 5, "hour": 14}
        )

        # Plot CTR trends
        models_data = {
            "PBM": pbm_probs,
            "Cascade": cascade_probs,
            "Enhanced PBM": enhanced_probs
        }
        plot_ctr_trends(models_data)

        test_data = [(positions, relevance_scores, [0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.05, 0.01])]

        metrics_data = {
            "PBM": evaluate_model(pbm, test_data),
            "Cascade": evaluate_model(cascade, test_data),
            "Enhanced PBM": evaluate_model(enhanced_pbm, test_data)
        }
        plot_metrics_comparison(metrics_data)
```




```
<ipython-input-6-4362a42e4f62>:146: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e.
  axes[i].set_xticklabels(models, rotation=45)
<ipython-input-6-4362a42e4f62>:146: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e.
  axes[i].set_xticklabels(models, rotation=45)
```